# CSci 231 Homework 6 – SOLUTIONS

Binary Search Trees and Hashing

CLRS Chapter 11.1-11.3 and 12.1-12.3

*Write and justify your answers on this sheet in the space provided.*[1]

1. (CLRS 12.2-5) Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

   **Solution:** Denote for a node $x$ having two children its predecessor by $p$ and its successor by $s$. First we show by contradiction that the sucessor of $x$ has no left child. Suppose $s$ has a left child. Then the key of $s$ is greater than that of $left[s]$. The key of $s$ is also larger than the key of $x$, and since $s$ has a left child the key of $left[s]$ is larger than that of $x$. Thus

   $$key[s] \geq key[left[s]] \geq key[x],$$

   which is a contradiction, since $s$ is the successor of $x$. Hence the sucessor of $x$ has no left child.

   Similarly, we show by contradiction that the predecessor of $x$ has no right child. Suppose $p$ has a right child. Then the key of $p$ is less than that of $right[p]$. The key of $p$ is also less than the key of $x$, and since $p$ has a right child the key of $right[p]$ is less than that of $x$. Thus

   $$key[p] \leq key[right[p]] \leq key[x],$$

   which is a contradiction, since $p$ is the predecessor of $x$. Hence the predecessor of $x$ has no right child.

2. (CLRS 11.2-2) Demonstrate the insertion of the keys $5, 28, 19, 15, 20, 33, 12, 17, 10$ into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \ mod \ 9$.

[1] Collaboration is allowed and encouraged, if it is constructive and helps you study better. Remember, exams will be individual. Write up the solutions on your own.

3. (CLRS 11.1-4) We wish to implement a dictionary by using direct addressing on a *huge* array. At the start, the array entries may contain garbage, and initializing the entire array is impractical because of its size. Describe a scheme for implementing a direct-address dictionary on a huge array. Each stored object should use $O(1)$ space; the operations SEARCH, INSERT and DELETE should take $O(1)$ time each; and the initialization of the data structure should take $O(1)$ time.

   *(Hint: Use an additional stack, whose size is the number of keys actually stored in the dictionary, to help determine whether a given entry in the huge array is valid or not.)*

   **Solution:** To implement a direct-address dictionary on a huge array, we initialize an additional stack which we can use to help determine whether a given entry in the huge array is valid or not. Call the arrays *huge* and *stack* and define a variable *head*, initially -1, to point to the top of the stack array. Then $head = -1$ corresponds to the dictionary being empty. Suppose we want to insert an element with key $x$. We increment *head*, set $huge[x] = head$ and $stack[head] = x$. Now suppose we want to SEARCH for an element in the huge array with key $y$. Then the following conditions must be satisfied:

   - $0 \leq huge[y] \leq head$,
   - $stack[huge[y]] = y$,

   Thus SEARCH and INSERT can be done in constant time. To DELETE an element with key $x$, we first SEARCH to make sure the element is in the huge array. In the stack array we have a position that will correspond to a deleted element, so we take the element at $stack[head]$ and copy it to $stack[huge[x]]$. We set then set $huge[stack[head]] = huge[x]$, $stack[head] = \text{NULL}$ and $huge[x] = \text{NULL}$ and decrement *head*. DELETE is also performed in constant time since it only involves a constant number of operations. The initialization of the data structure is done in constant time since the allocation of the stack takes constant time.